11 different ways for Outlier Detection in Python

Outlier Detection Python is a specialized task which has various use-cases in Machine Learning. Use-cases would be anomaly detection, fraud detection, outlier detection etc. There are many ways we can find outliers in your analysis.

So, let us talk about outliers in your datasets and explore various quick ways we can identify outliers in daily analytics lifecycle.

In this post, you will learn :

- What are outliers?
- How do outliers occur in your dataset
- What are the simple methods to identify outliers in your dataset
- Outliers Detection using machine learning algorithms – Robust Covariance, One-Class SVM, Isolation Forest, Local Outlier Factor

What are outliers?

Outliers: in simple terms outliers are data points which are significantly different from your entire datasets. For e.g.

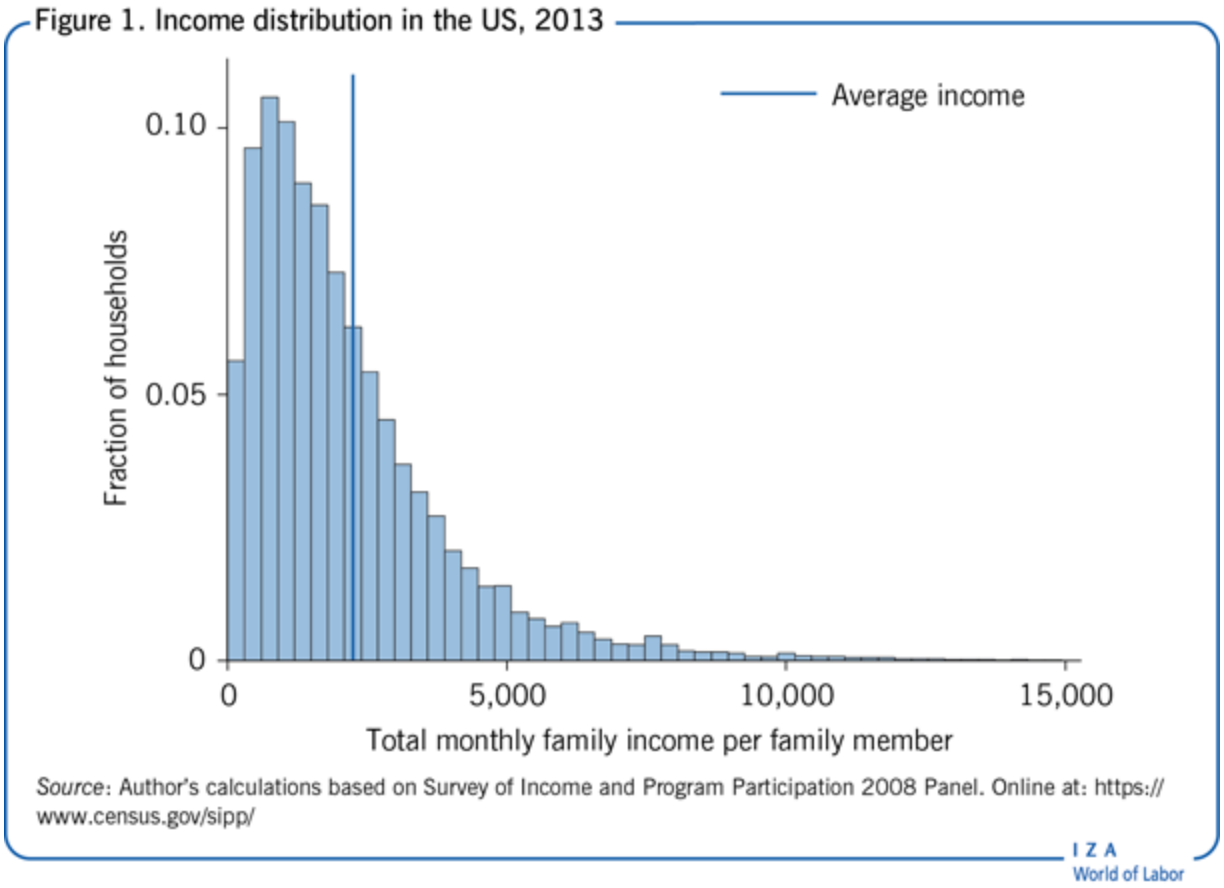


Image source

How do outliers occur in a datasets?

Outliers occur either by chance, or either by measurement error or data population is heavy tailed distribution as shown above.

Main effects of having outliers are that they can skew your analytics in poor analysis, longer training time and bad results at the end. Most importantly, this distorts the reality which exists in the data.

Simple methods to Identify outliers in your datasets.

Sorting – If you have dataset you can quickly just sort ascending or descending.

While it is looks so obvious, but sorting actually works on real world.

Outlier Detection Python – Quick Method in Pandas – Describe( ) API

```
import numpy as np
import pandas as pd

url =
'https://raw.githubusercontent.com/Sketchjar/MachineLearningHD

df = pd.read_csv(url)
df.describe()
```

Income (Million)
1.5895
1.6508
1.7131
1.7136
1.7212
1.7296
1.7343
1.7663
1.8018
1.8394
1.8869
1.9357
1.9482
2.1038
10.8135

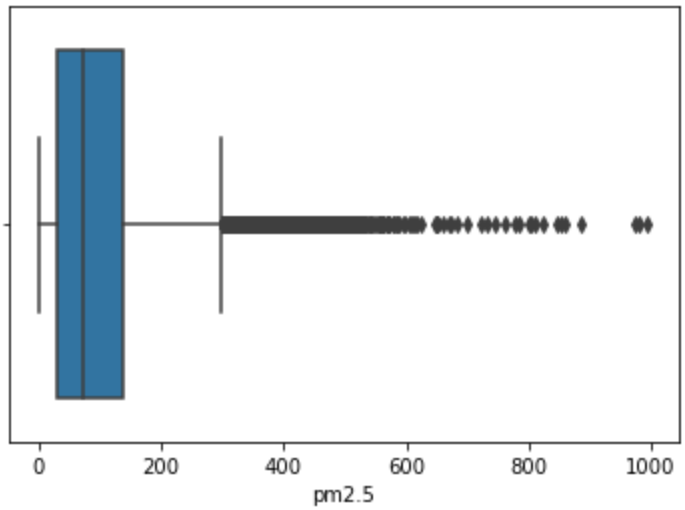
	year	month	day	hour	pm2.5	DEWP	TEMP	PRES	Iws	Is
count	43824.000000	43824.000000	43824.000000	43824.000000	41757.000000	43824.000000	43824.000000	43824.000000	43824.000000	43824.000000
mean	2012.000000	6.523549	15.727820	11.500000	98.613215	1.817246	12.448521	1016.447654	23.889140	0.052734
std	1.413842	3.448572	8.799425	6.922266	92.050387	14.433440	12.198613	10.268698	50.010635	0.760375
min	2010.000000	1.000000	1.000000	0.000000	0.000000	-40.000000	-19.000000	991.000000	0.450000	0.000000
25%	2011.000000	4.000000	8.000000	5.750000	29.000000	-10.000000	2.000000	1008.000000	1.790000	0.000000
50%	2012.000000	7.000000	16.000000	11.500000	72.000000	2.000000	14.000000	1016.000000	5.370000	0.000000
75%	2013.000000	10.000000	23.000000	17.250000	137.000000	15.000000	23.000000	1025.000000	21.910000	0.000000
max	2014.000000	12.000000	31.000000	23.000000	994.000000	28.000000	42.000000	1046.000000	585.600000	27.000000

If you see in the pandas dataframe above, we can quick visualize outliers. For e.g. in pm2.5 column maximum value is 994, whereas mean is only 98.613.

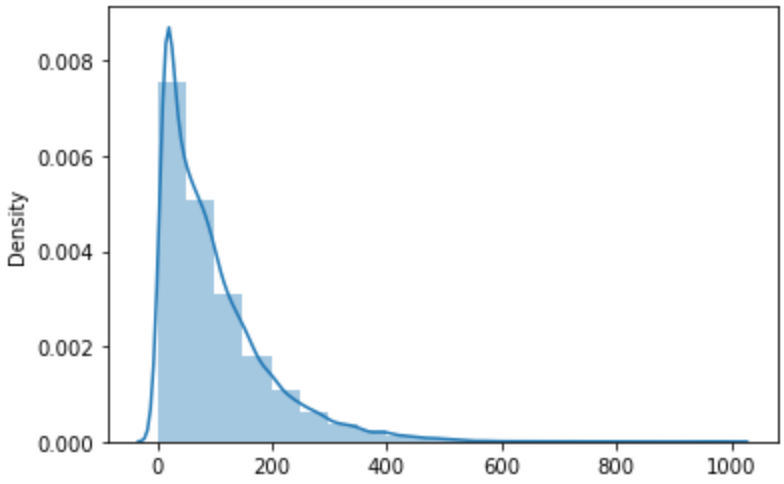
Data Visualization using Box plots, Histograms, Scatter plots

If we plot a **boxplot** for above pm2.5, we can visually identify outliers in the same.

Histograms



BoxPlot to visually identify outliers



Again similar data but different visualization, we can see that there are some long tail outliers in the data.

Outlier Detection using Z-Scores

Z-scores can help in finding unusual data points with our datasets when our data is following normal distribution.

Z score formula is **(X – mean)/Standard Deviation**

We can see outlier at the bottom of the table has different Z-Score as compared to others.

Income (Million)	Z-score
1.5895	-0.3460
1.6508	-0.3198
1.7131	-0.2930
1.7136	-0.2928
1.7212	-0.2895
1.7296	-0.2859
1.7343	-0.2839
1.7663	-0.2702
1.8018	-0.2550
1.8394	-0.2389
1.8869	-0.2185
1.9357	-0.1976
1.9482	-0.1922
2.1038	-0.1255
10.8135	3.6091

Create outlier Fences using Interquartile Range

IQR is basically middle of our dataset which is also know as Median of a dataset. We can calculate IQR with following formula (Q3- Q1).

Now based on IQR we can assign lower outer, lower inner, upper inner, and upper outer ranges and all the data points which are outside this range can be considered as outliers.

Hypothesis Testing – Grubb's Outlier test

Grubb's outlier test can only detect uni variate outliers, however there are other tests which are available like Tietjen-Moore test. However it requires specific number of outliers, which is difficult to do as we are trying to find out the outliers in first place.

Outlier Detection Using Machine Learning

In this section , we will discuss four machine learning techniques which you can use for outlier detection.

Robust Covariance – Elliptic Envelope

This method is based on premises that outliers in a data leads increase in covariance, making the range of data larger. Subsequently the determinant of covariance will also increase, this in theory should reduce by removing the outliers in the datasets. This method assumes that some of hyper parameters in n samples follow Gaussian distribution. Here is flow on how this works:

1.Repeat k times:

1. Sample Points randomly and compute there mean and covariance
    1. Repeat it twice:

1.2.1 Compute mahalonobis distances for all points and sort them in ascending order

1.2.2 Use smallest hyper parameter distances to computer new estimates of mean and covariance

2. Find the determinant of covariance

2.1 Repeat the step again with small subset until convergence which means determinants are equal

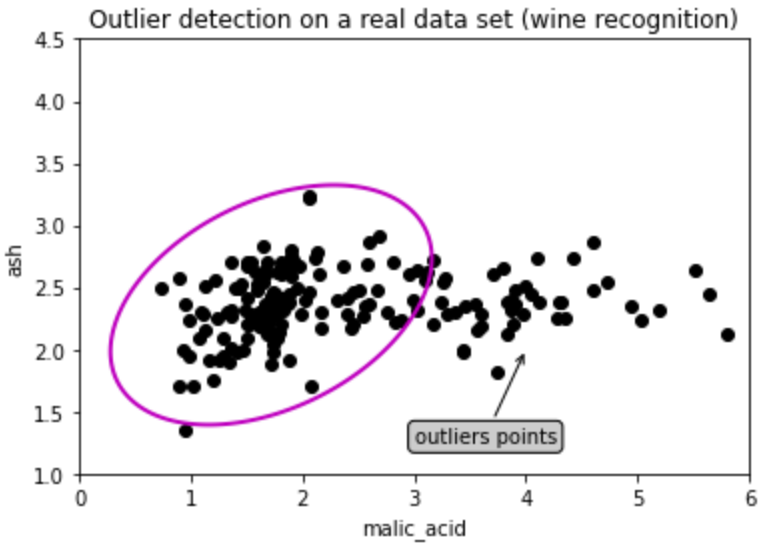
2.2 Repeat all points in 1(a) and 1(b)

3. In all subsets of data, **use the estimation of smallest determinant and find mean and covariance.**

More information on theory about Robust covariance. Here is a link

Outlier Detection Python Code – Elliptic Envelope

```
import numpy as np
from sklearn.covariance import EllipticEnvelope
import matplotlib.pyplot as plt
import matplotlib.font_manager
from sklearn.datasets import load_wine
%matplotlib inline

# Define "classifiers" to be used
classifiers = {
        "Robust Covariance (Minimum Covariance Determinant)":
    EllipticEnvelope(contamination=0.25),
    }
colors = ['m', 'g', 'b']
legend1 = {}
legend2 = {}

# Get data
X1 = load_wine()['data'][:, [1, 2]]  # two clusters

# Learn a frontier for outlier detection with several classifiers
xx1, yy1 = np.meshgrid(np.linspace(0, 6, 500), np.linspace(1, 4.5, 500))
for i, (clf_name, clf) in enumerate(classifiers.items()):
    plt.figure(1)
    clf.fit(X1)
    Z1 = clf.decision_function(np.c_[xx1.ravel(), yy1.ravel()])
    Z1 = Z1.reshape(xx1.shape)
    legend1[clf_name] = plt.contour(
        xx1, yy1, Z1, levels=[0], linewidths=2, colors=colors[i])

legend1_values_list = list(legend1.values())
legend1_keys_list = list(legend1.keys())

# Plot the results (= shape of the data points cloud)
plt.figure(1)  # two clusters
plt.title("Outlier detection on a real data set (wine recognition)")
plt.scatter(X1[:, 0], X1[:, 1], color='black')
bbox_args = dict(boxstyle="round", fc="0.8")
arrow_args = dict(arrowstyle="->")
plt.annotate("outliers points", xy=(4, 2),
             xycoords="data", textcoords="data",
             xytext=(3, 1.25), bbox=bbox_args, arrowprops=arrow_args)
plt.xlim((xx1.min(), xx1.max()))
plt.ylim((yy1.min(), yy1.max()))

plt.ylabel("ash")
plt.xlabel("malic_acid")

plt.show()
```



Code Source: Scikit Learn

One-Class SVM

One class Support Vector Machine is a special case in support vector machines which is used for unsupervised outlier detection. For more information on support vector, please visit this link.

Let see outlier detection python code using One Class SVM. We will see two different examples for it.

```
from sklearn.svm import OneClassSVM
X = [[0], [0.44], [0.45], [0.46], [1]]
clf = OneClassSVM(gamma='auto').fit(X)
clf.predict(X)

array([-1,  1,  1,  1, -1, -1, -1], dtype=int64)
Here -1 refers to outlier and 1 refers to not an outliers.
```

Let us see another example

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.font_manager
from sklearn import svm

xx, yy = np.meshgrid(np.linspace(-5, 5, 500), np.linspace(-5, 5, 500))
# Generate train data
X = 0.3 * np.random.randn(100, 2)
X_train = np.r_[X + 2, X - 2]
# Generate some regular novel observations
X = 0.3 * np.random.randn(20, 2)
X_test = np.r_[X + 2, X - 2]
# Generate some abnormal novel observations
X_outliers = np.random.uniform(low=-4, high=4, size=(20, 2))

# fit the model
clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
clf.fit(X_train)
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_outliers = clf.predict(X_outliers)
n_error_train = y_pred_train[y_pred_train == -1].size
n_error_test = y_pred_test[y_pred_test == -1].size
n_error_outliers = y_pred_outliers[y_pred_outliers == 1].size

# plot the line, the points, and the nearest vectors to the plane
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.title("Outlier Detection")
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), 0, 7), cmap=plt.cm.PuBu)
a = plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='darkred')
plt.contourf(xx, yy, Z, levels=[0, Z.max()], colors='palevioletred')

s = 40
b1 = plt.scatter(X_train[:, 0], X_train[:, 1], c='white', s=s, edgecolors='k')
b2 = plt.scatter(X_test[:, 0], X_test[:, 1], c='blueviolet', s=s,
                 edgecolors='k')
c = plt.scatter(X_outliers[:, 0], X_outliers[:, 1], c='gold', s=s,
                edgecolors='k')
plt.axis('tight')
plt.xlim((-5, 5))
plt.ylim((-5, 5))
plt.legend([a.collections[0], b1, b2, c],
           ["learned frontier", "training observations",
            "new regular observations", "new abnormal observations"],
           loc="upper left",
           prop=matplotlib.font_manager.FontProperties(size=11))
plt.xlabel(
    "error train: %d/200 ; errors novel regular: %d/40 ; "
    "errors novel abnormal: %d/40"
    % (n_error_train, n_error_test, n_error_outliers))
plt.show()
```

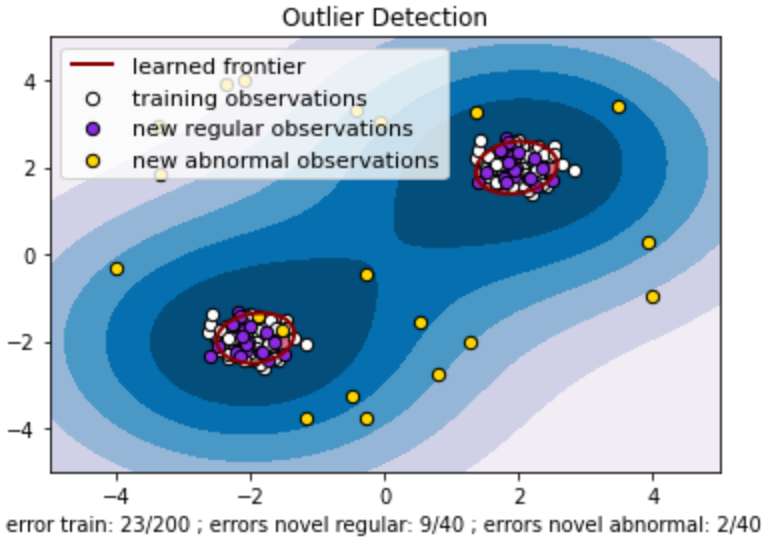


Image and code source: Scikit Learn

Isolation Forest

Isolation Forest is an ensemble model which isolates observations by randomly selecting a feature and selecting a split value between maximum and minimum of selected feature.

Since this recursive partitioning is represented by a tree structure, and number of splittings is equivalent to path length from root node to terminating node. For more information, use this link.

See Isolation Forest in code.

```
from sklearn.ensemble import IsolationForest
X = [[-1.1], [0.3], [0.5], [100]]
clf = IsolationForest(random_state=0).fit(X)
clf.predict([[0.1], [0], [90]])

array([ 1,  1, -1])
Here -1 refers to outlier and 1 refers to not an outliers.
```

Local Outlier Factor (LOF)

LOF computes local density deviation of a certain point as compared to its neighbors. It is different variant of k Nearest neighbors. Simply, in LOF outliers is considered to be points which have lower density than its neighbors.

Local Outlier Factor in Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import LocalOutlierFactor

print(__doc__)

np.random.seed(42)

# Generate train data
X_inliers = 0.3 * np.random.randn(100, 2)
X_inliers = np.r_[X_inliers + 2, X_inliers - 2]

# Generate some outliers
X_outliers = np.random.uniform(low=-4, high=4, size=(20, 2))
X = np.r_[X_inliers, X_outliers]

n_outliers = len(X_outliers)
ground_truth = np.ones(len(X), dtype=int)
ground_truth[-n_outliers:] = -1

# fit the model for outlier detection (default)
clf = LocalOutlierFactor(n_neighbors=20, contamination=0.1)
# use fit_predict to compute the predicted labels of the training samples
# (when LOF is used for outlier detection, the estimator has no predict,
# decision_function and score_samples methods).
y_pred = clf.fit_predict(X)
n_errors = (y_pred != ground_truth).sum()
X_scores = clf.negative_outlier_factor_

plt.title("Local Outlier Factor (LOF)")
plt.scatter(X[:, 0], X[:, 1], color='k', s=3., label='Data points')
# plot circles with radius proportional to the outlier scores
radius = (X_scores.max() - X_scores) / (X_scores.max() - X_scores.min())
plt.scatter(X[:, 0], X[:, 1], s=1000 * radius, edgecolors='r',
            facecolors='none', label='Outlier scores')
plt.axis('tight')
plt.xlim((-5, 5))
plt.ylim((-5, 5))
plt.xlabel("prediction errors: %d" % (n_errors))
legend = plt.legend(loc='upper left')
legend.legendHandles[0]._sizes = [10]
legend.legendHandles[1]._sizes = [20]
plt.show()
```

In Summary , we have discussed various quick methods through we can identify outliers. There are other advanced machine learning models which can also be used to identify outliers, however we will discuss them in a separate post.

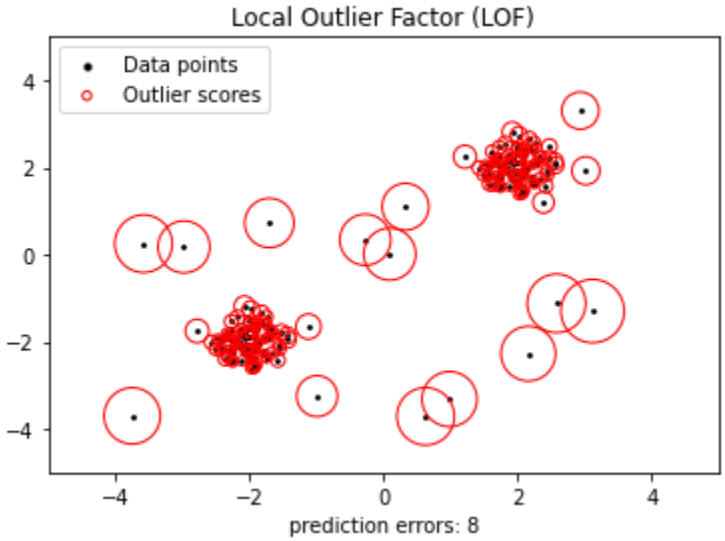


Image and code source: Scikit Learn

In summary, we have discussed various ways eleven different ways for detecting outliers using Python.

- seven different ways to detect outliers by visualization, statistics
- four different ways to detect outliers by machine learning model